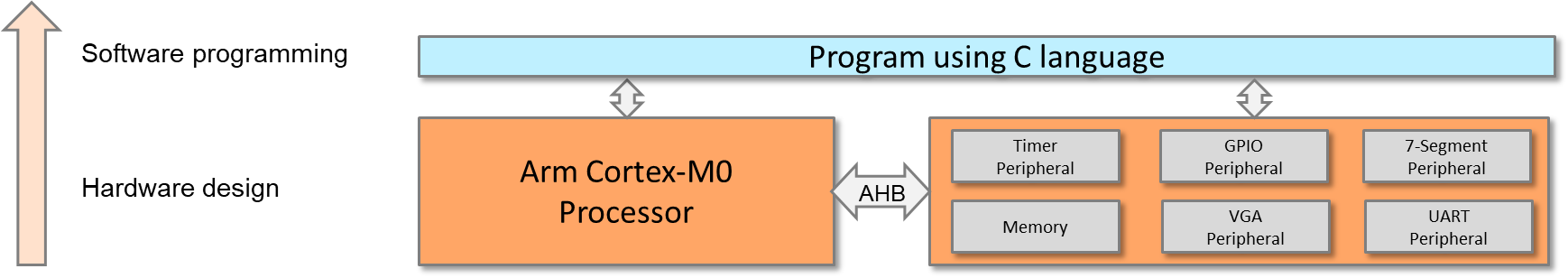
Lab Exercise 9: Programming an SoC Using C Language

# Overview

In the previous labs, assembly language was used to program the SoC at low-level, which gave better control over the hardware and helped us understand the hardware mechanism at register transfer level (RTL).

In this lab, C language will be used to program the SoC at a higher level, whereby compilers, software libraries, and tools can be used to hide the complexity of low-level management. This means that we can make more complicated applications in a more efficient way.



**Program SoC using C**

One can always mix C and assembly code in one project, or even in one file.

Hardware implementation has been completed in the previous labs; the following labs will focus on software. The work in this lab includes:

* Software programming:
* Program the Cortex-M0 processor and control the all peripherals using C language.
* Demonstrate the SoC:
* Use the timer interrupt to implement a counting up counter (counting from 0 to 9) and display the value to the VGA display (same as previous lab, but using C programing).
* Use the UART interrupt to send characters to a PC or laptop (same as previous lab, but using C programing).
* Input from 8-bit switch and output to LEDs

# Details

## software

The files that will be used are listed below:

FILES TO BE USED

|  |  |
| --- | --- |
| **File name** | **Description** |
| cm0dsasm.s | The assembly file that includes interrupt vectors and part of the interrupt service routines |
| main.c | The C code that contains the main program and part of the interrupt service routines |

### program procedure

The assembly code in cm0dsasm.s should perform the following:

* Initialize the interrupt vector.
* Define heap and stacks.
* Reset handler.
  + Branch to the main code in main.c.
* Timer handler
  + Push registers (e.g., R1 – R4) to the stack.
  + Branch to the timer interrupt service routine in main.c.
  + Pop registers from the stack.
* UART handler
  + Push registers (e.g., R1 – R4) to the stack.
  + Branch to the UART interrupt service routine in main.c.
  + Pop registers from the stack.

The C code in main.c should perform the following:

* Main program
  + Set the timer interrupt priority to 0x00 (higher).
  + Set the UART interrupt priority to 0x40 (lower).
  + Enable interrupts for the timer and UART.
  + Reset the 7-segment display.
  + Initialize the timer to generate an interrupt every second.
    - Write the load value register, e.g., 50,000,000.
    - Set prescaler, e.g., 1x or 16x.
    - Change the operation mode to load mode.
  + Start the timer.
  + Repeat the following:
    - Change the GPIO to read mode.
    - Read from the switches using the GPIO.
    - Change the GPIO to write mode.
    - Write to the LEDs.
* Timer interrupt handler
  + Clear the timer interrupt request.
  + Increment the counter.
  + Display the counter to the VGA text region.
  + Disable the timer interrupt if the counter reaches 9.
* UART interrupt handler
  + Read from the UART (from the keyboard).
  + Write to the UART (to the terminal window).

An example of the demo:



**Demo Example**

# extension work

Here are some extra things that you can do:

* Optimize the code in both operational speed and code density. You can use:
  + Embedded assembly code in C code
  + Embedded C code in assembly code
* Evaluate the coding efficiency between handwritten assembly code and assembly generated by the compiler.